# *Adreßrechnung im Überblick*

## *Weshalb Adreßrechnung?*

- zum Zugriff auf Komponenten heterogener Datenstrukturen,
- zum Zugriff auf Komponenten homogener Datenstrukturen,
- zum Zugriff auf lokale und globale Variable,
- um die fortlaufende Adressierung in Programmschleifen zu unterstützen,
- um die Verschieblichkeit von Datenstrukturen und Programmen zu unterstützen,
- um Mehrfachverzweigungen zu unterstützen.

## *Adreßmodelle*

**1. Absolutadressierung**
Adresse ist Direktwert im Befehl.

- einfachste Lösung.
- schnell, da Adreßrechnung entfällt.
- Adreßrechnung nur durch Befehlsmodifikation.

**2. Indirekte Adressierung (Adresse von Adresse)**
Adresse wird aus Speicherzelle entnommen, die ihrerseits mittels Direktwert im Befehl adressiert wird.

*Mehrstufige indirekte Adressierung*
Eine gegebene Adresse adressiert eine Speicherzelle, die ihrerseits eine Adresse enthält usw. Die zuletzt geholte Adresse adressiert die Datenstruktur.

Woher weiß man, daß es sich um die letzte Adresse handelt?

- durch Angabe der Stufenzahl (Levels of Indirection),
- durch eine Endekennung in der Adreßangabe (wenn Adreßlänge < Wortlänge; z. B. MSB = 0: Adresse von Adresse, MSB = 1: Datenadresse).

**3. Registeradressierung**
Adresse wird aus programmseitig ladbarem Register entnommen

- universell,
- technisch einfach,
- gibt es an sich nur wenige Register, so verringert sich die Zahl der eigentlichen Operandenregister noch weiter.
- die weitaus meisten Speicherzugriffe erfordern eigens Adressierungsbefehle.

**4. Verbundlösungen aus Register- und Befehlsadressierung**
Die einfachste Form: Basis + Displacement.

**Register oder Verbundadressierung?**

*Die herkömmliche Auslegung:*
Basis + Displacement oder komfortabler:

- wenn man nur wenige Register hat, kann man nicht allzu viele zu Adressierungszwecken beiseite setzen,
- der einzelne Befehl - mühsam aus dem Speicher geholt - soll möglichst viel leisten (CISC-Philosophie),
- die lokalen Variablen eines Funktionsaufrufs passen nicht (oder nicht vollständig) in Register, so daß immer wieder Zugriffe auf den Stack Frame im RAM nötig sind.

*RISC-Philosophie:*
Basis + Displacement mit Displacements von ca. 13...16 Bits (14 reichen meistens). Auf dieser Grundlage wird alles andere erledigt (z. B. die reine Registeradressierung mitttels Displacement = 0).

*Eine neumodische Auslegung (IA-64):*
Nur Registeradressierung:

- es gibt genügend Register (128 Stück),
- die lokalen Variablen eines Funktionsaufrufs werden typischerweise im Registerstack bereitgestellt. Die Speicheradressierung dient deshalb häufig nur zu Blocktransporten (Variable rein/raus) und zu fortlaufenden Zugriffen in Programmschleifen.

**5. Segmentierung**
Stellt unabhängige Adreßräume bereit, die jeweils von Adresse 0 an beginnen. Unterstützt Verschieblichkeit von Datenstrukturen und Programmen.

Konsequentes Ausnutzen von Segmentierungsvorkehrungen erlaubt es, in vielen Fällen mit Absolutadressen auszukommen. Verwaltungsaufwand hoch, Hardwareaufwand hoch (vgl. IA-32). Adreßrechnung nicht überflüssig (wird zum Zugreifen auf Komponenrten von Datenstrukturen nach wie vor gebraucht).

**6. Objektorientierung**
Programmierer sieht keine Adressen, sondern nur Ordinalzahlen der Objekte und ihrer Komponenten. Beispiele: Burroughs B 5500...6800, iAPX-432 (Intel), AS/400 (IBM). Elegant. Hoher Laufzeit-Overhead (1 Operandenzugriff erfordert typischerweise wenigstens 2 Tabellenzugriffe). Läuft nur zufriedenstellend mit massiver Hardware-Unterstützung (Krankheit des iAPX-432). IA-32-Segmentierung ließe sich in diesem Sinne ausnutzen. Macht aber keiner...